Implementing a safe embedded computing system in SRAM-based FPGAs using IP cores: A case study based on the Altera NIOS-II soft processor

Implementing a safe embedded computing system in SRAM-based FPGAs using IP cores:

a case study based on the Altera NIOS-II soft processor

Julio Perez Acle
Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
julio@fing.edu.uy

Matteo Sonza Reorda, Massimo Violante
Dip. Automatica e Informatica
Politecnico di Torino
Torino, Italy
{matteo.sonzareorda, massimo.violante}@polito.it

***Abstract*— Reconfigurable Field Programmable Gate Arrays (FPGAs) are growing the attention of developers of mission- and safety-critical applications (e.g., aerospace ones), as they allow unprecedented levels of performance, which are making these devices particularly attractive as ASICs replacement, and as they offer the unique feature of in-the-field reconfiguration. However, the sensitivity of reconfigurable FPGAs to ionizing radiation mandates the adoption of fault tolerant mitigation techniques that may impact heavily the FPGA resource usage. In this paper we consider time redundancy, that allows avoiding the high overhead that more traditional approaches like N-modular redundancy introduce, at an affordable cost in terms of application execution-time overhead. A single processor executes two instances of the same software sequentially; the two instances are segregated in their own memory space through a soft IP core that monitors the processor/memory interface for any violations. Moreover, the IP core checks for any processor functional interruption by means of a watchdog timer. Fault injection results are reported showing the characteristics of the proposed approach.**

Keywords- Embedded systems; Fault Tolerance; FPGA; IP cores; Fault Injection

I. Introduction

Today reprogrammable Field Programmable Gate Arrays (FPGAs) are increasingly attracting the attention of developers of safety- and mission-critical applications (e.g., in the aerospace domain) for a number of reasons. First of all, modern FPGA devices offer an unprecedented level of resources (logic, memory, interconnection, arithmetic, and processing resources) that make them highly competitive with ASICs in markets where low production volumes and short time to market are crucial. Secondly, reprogrammable FPGAs offer a competitive advantage with respect to ASICs in being reconfigurable: when deployed in the field (i.e., in a satellite already in orbit) their configuration can be changed to improve the functionalities they provide (e.g., to change a baseband processing algorithm in a software defined radio system), to correct bugs, to adapt to changing environment conditions, or to implement reconfigurable computing architectures.

Two main technologies are nowadays available for implementing reconfigurable FPGAs: one based on an SRAM configuration memory, where the information defining what functions the FPGA implements is stored on-chip using SRAM memory bits (e.g., Xilinx Virtex devices, and Altera Cyclone devices), and one based on a Flash configuration memory, where the configuration is stored in floating gate cells (e.g., Actel ProASIC devices). When deploying reprogrammable devices in a radioactive environment (such as the space one), particular care must be posed to the phenomena induced by ionizing radiations, which may impair some functionalities of the circuit the FPGA implements, or even the whole device. In case of SRAM-based FPGAs, ionizing radiation may induce modifications to the configuration information, provoking Single Event Upsets (SEUs) that remain latched until a fresh image of the configuration memory is restored. Conversely, Flash-based FPGAs are immune to configuration memory SEUs. Both SRAM-based and Flash-based FPGAs suffer from radiation-induced SEUs in the memory elements hosted by the reconfigurable fabric (flip-flops, and memory arrays). Moreover, in the case radiations hit the FPGA control logic, effects known as Single Event Functional Interruptions (SEFIs) may be observed, which impair the correct operation of the FPGA until a reset, or a power cycle is performed [1][2].

To cope with SEUs and SEFIs designers must employ radiation mitigation techniques, which consist in introducing some sort of redundancy in the implemented design or system. The most widely used approach is Triple Modular Redundancy (TMR) [3][4] that mandates to replicate three times the design (only its memory elements in case of Flash-based FPGAs, or the entire design in case of SRAM-based ones, or even the entire FPGA chip in case of system-level mitigation) and to add majority voters. In case of processor cores implemented on FPGAs, alternative approaches can be exploited to save FPGA resources at the cost of increased computing time. In [5], an approach is presented where the processor core is duplicated, and a custom IP core manages the concurrent execution of the same application on the two cores that work synchronously. Although this approach is effective in reducing the resource

This work was partially supported by a grant from the National Agency of Research and Innovation (ANII) from Uruguay.

overhead with respect to TMR, it still requires processor duplication.

In this work we investigate the possibility to further reduce the area overhead by exploiting time redundancy [3]. According to time redundancy, the same application is executed twice (three times in case we want to achieve error masking), and then an acceptance test is executed. If the two results match, one of the two is forwarded to the user; otherwise, the outputs are discarded, and the computation repeated. An example of application of this concept to processor-based systems for space applications can be found in [6], where the tasks executed by a processor are duplicated, and executed in segregation: each task can access only to its own memory. A custom companion chip takes care of managing any memory access, and guarantees the task segregation.

In this paper we exploit the concept of time redundancy in developing an architecture inspired to DMT [6] using a soft processor core aiming at being implemented in an SRAM-based FPGA. In order to enforce task segregation, as well as protection against possible SEFIs of the processor core, we resorted to the processor Memory Protection Unit available as soft IP for the selected processor core, and developed two additional IP cores implementing a watchdog timer and a DMA controller.

The main contribution of this paper lies in the experimental validation of the feasibility of achieving a safe system by using a mix of already available and ad hoc developed (and highly reusable) IP cores, thus minimizing the development time. In our work we exploited the Altera NIOS-II [7] as processor core, and the Altera Memory Protection Unit IP core for memory segregation. Being all the cores already available, and validated, the design effort is limited to the integration with a custom watchdog timer and DMA controller. By exploiting already existing cores a robust system can be obtained, which can be used with a number of different FPGAs supporting the same cores. As a result, a general architecture is obtained which is highly portable and reusable.

To assess the effectiveness of the proposed approach, we also performed a set of fault injection experiments, which show how the SEUs and SEFIs that may affect the processor are effectively mitigated by our architecture.

The rest of the paper is organized as follows. Section II. presents the adopted architecture. Section III. describes the experiments we performed to assess the soundness of the architecture we developed. Section IV. draws some conclusions and outlines future works.

II. Adopted architecture

This section describes the architecture we adopted, and details its implementation. The architecture is intended for hardening computing intensive applications executed by a commercial-off-the-shelf processor implemented on SRAM-based FPGAs. The application is supposed to entail a data acquisition phase during which an input buffer is filled with the data that has to be processed, followed by a data processing phase during which an algorithm is applied over the input data and an output buffer is produced, and finally a data presentation phase during which the output data is delivered to the user.

A. Overview

The architecture focuses on providing protection against transient errors induced by ionizing radiation, the so called Single Event Upsets (SEUs), which may affect the execution of the application by altering the content of processor registers, or by altering the configuration memory of the FPGA. Two protection mechanisms are used.

As far as SEUs affecting the processor memory elements (i.e., register file, special purpose registers, and cache memory) are concerned, task-level duplication is exploited in combination with hardware-assisted consistency check. The application is executed twice, each time writing to different memory locations; at the end, the two output buffers are compared. In case of mismatch, the processor undergoes a reset operation, and the whole process is repeated. In case of successful match, the data presentation phase issues to the user the two output buffers: the two buffers are sent to provide a protection mechanism against possible data-transfer errors. In order to guarantee that the two executions of the application instances are performed independently, so that any SEU may affect one and only one of the two executions, a special-purpose hardware module, called *smart watchdog*, is used. The smart watchdog is implemented in the same FPGA used for implementing the processor, and it is placed on the process bus between the processor and the memory hierarchy. It has indeed to snoop for any memory access directed either toward the cache or the main memory. Moreover, the smart watchdog must be able to access to the memory space of the processor to perform the consistency check of the two output buffers.

Let us call I1 and I2 the two instances of the application that are executed sequentially. The processor memory is partitioned in two regions, R1 associated to I1, and R2 associated to I2. Each instance Ix is allowed to read/write only within the boundary of Rx; any access outside Rx indicates the occurrence of an error. The smart watchdog is in charge of monitoring any access the processor performs, and in case Ix is accessing to an address in Ry with $x \neq y$ a non-maskable interrupt resulting in the processor reset is activated.

The sequence of operations performed by the processor during the execution of an application is the following:

1) The processor programs the smart watchdog to define R1 and R2, and resets the watchdog timer of the smart watchdog.

2) The processor selects R1 and initiates the execution of I1.

3) Upon completion of I1, the processor resets the watchdog timer.

4) The processor selects R2 and initiates the execution of I2.

5) Upon completion of I2, the processor resets the watchdog timer

6) The smart watchdog performs the consistency check.

7) The processor resets the watchdog timer and repeats from (2).

Any operation resulting in a wrong memory access outside the region associated to an instance of the application triggers a non-maskable interrupt leading to processor reset. Moreover, any operation leading to a processor hang (i.e., a SEFI) leads to the watchdog expiration, which triggers the processor reset as well. The smart watchdog is in charge of the consistency check: it performs a word-by-word comparison of the output buffers computed by the two application instances. In case of mismatch the processor is reset, otherwise the whole process is repeated. The comparison is implemented through DMA burst transfers that read and compare the two output memory buffers.

As far as SEUs in the FPGA configuration memory are concerned, on-line checking is performed: the configuration memory is constantly read, a checksum is computed and compared with a known-good value. In case of mismatch, the FPGA device is reset, a fresh image of the configuration memory written to the device, and the whole application started from scratch.

B. Implementation

We developed a proof-of-concept implementation of the described architecture on an Altera Cyclone-II device, using the NIOS-II processor core. For the sake of this paper we focused only on SEUs affecting the processor. The features provided by Altera devices can be straightforwardly exploited to implement the protection mechanism against SEUs in the device configuration memory.

The NIOS-II is a 32 bit, 6-stage pipeline RISC processor. Options include separated instruction and data cache, and a full Memory Management Unit (MMU) or a simpler Memory Protection Unit (MPU). The MPU has separated instruction and data regions. Execution permission can be granted on instruction regions and read or read/write permissions on data regions, both for user and supervisor execution mode. An exception is raised in case of permission violation. Additional exception conditions relevant from a safety point of view are misaligned memory accesses and illegal instruction opcode. The MPU, which is available as IP core, is the building block of our smart watchdog, as described in II.A..

We developed a system encompassing one NIOS-II (version f) core with a 4 Kbytes instruction cache and a 2KBytes data cache. A smart watchdog is attached to the processor encompassing the MPU configured with 6 data regions and 4 instruction regions, an interval timer used as watchdog timer, and a simple DMA controller for output buffer comparison. 512KBytes of main memory are attached to the processor, implemented outside the FPGA device using SRAM chips. Being based on a combination of already existing IP cores, and some custom-made modules, the implementation of the system is highly portable, and it can be mapped on any FPGA device supporting the NIOS-II processor and its MPU. In our implementation we considered Cyclone-II devices, obtaining the resource occupation figures of TABLE 1.

TABLE I. RESOURCE OCCUPATION

Module	Logic resources [#]	Flip-flops [#]	Memory bits [#]
NIOS-II	2.063	1.549	63.104
MPU	963	729	256
Interval timer and DMA controller	350	256	128
TOTAL	3.376	2.534	63.488

With respect to a system including only the NIOS-II processor, the implementation of the proposed architecture leads to the following overheads (due to the addition of the MPU and smart watchdog): 63% of logic resources, 64% of flip-flops, and 0.6% of memory bits. As far as the application execution time is concerned, the proposed architecture introduces an overhead of 100%, as two instances of the application have to be executed sequentially. In case the NIOS-II system is implemented using TMR [3], limiting our analysis to the logic resources and flip-flops, we can expect logic resource and flip-flop overheads of at least 200% (not including the resources needed for implementing the majority voters), and at least a 15% in performance overhead due to majority voters added on the processor critical paths. These figures are expected to be even higher in case mitigation techniques for the cache system are considered where the implementation of a protection scheme like EDAC is expected to impact heavily on the memory bit occupation, logic/flip-flop resources and performance. In the case of the approach presented in [5], we can estimate a logic resource and flip-flop overhead of at least 100%, and a performance overhead of about 30%. Therefore, we can state that the adopted approach, when compared to alternative ones, is effective in reducing the FPGA resource overhead, at a cost of a higher application execution time.

III. EXPERIMENTAL RESULTS

To assess the robustness of the proposed architecture, we developed a fault injection system, and we performed a set of fault injection experiments. For the sake of this paper, we focused only on SEUs affecting the processor memory elements.

We used a software-based fault injection mechanism: we added a SEU injection routine to the code running on the processor, and we used an interrupt request to trigger the injection routine. The mechanism allows injections of SEUs in any software-accessible location within the processor, including general-purpose registers, control registers and program counter. SEUs are randomly injected both in time and space: the injection routine is activated in a randomly-selected clock cycle during application execution, and a SEU is injected in one randomly selected bit of a randomly selected register. The fault injection process encompasses the following operations:

1) The FPGA is configured, a fresh image of the processor memory is downloaded, and the processor is reset. Except for the FPGA configuration, this initialization

operation is performed for each fault so to guarantee that experiments are independent from each other.

2) Through the debug interface the memory location storing the injection time, and the injection location (register and bitmask to use) are modified according to the fault to be injected.

3) A timer initially set to the injection time is started, and the application execution is started according to the mechanism described in II.B..

4) Upon expiration of the injection-time timer, the application is stopped, and the SEU injection routine is activated; as a result, the desired fault is inoculated in the system.

5) The execution of the application is resumed until its completion.

At the end of the application execution, the two output buffers are analyzed, and SEUs are classified as follows:

- *No effect*: The execution completed successfully, the two output buffers contain the same values and match the results produced by a fault-free execution. Moreover, the MPU did not trigger the non-maskable interrupt, and the watchdog timer did not expire.
- *Data detection*: The execution completed successfully, but the output buffers produced by the two application instances have different values, while the smart watchdog signaled the mismatch.
- *Exception*: The smart watchdog triggered the non-maskable interrupt, signaling that an attempt to access a forbidden memory partition is detected.
- *Timeout*: The smart watchdog timer exhausted before the end of application execution.
- *Trap*: The injected fault triggered one of the processor traps, indicating that an erroneous situation is detected (e.g., the processor is executing a misaligned memory access).
- *Wrong answer*: The execution completed successfully, the two output buffers are equal but they do not match those produced by a fault-free execution. Moreover, the MPU did not trigger the non-maskable interrupt, and the watchdog timer did not expire. This condition corresponds to the case, in which the fault produced a misbehavior, but escaped all the error detection mechanisms our architecture offers.

In our experiments we considered a data-processing benchmark composed of a 16-tap finite impulse response filter processing 512 samples of a 500Hz tone sampled at 8KHz. The filter is implemented in the direct form, and its coefficients are chosen to obtain a 1KHz low pass filter. Then, a further processing based on computing the square root of the obtained values is executed. Benchmark execution, according to the sequence of operations described in Section II.B. lasts for about 8 million clock cycles.

During preliminary injection experiments to tune the system, some faults were identified that put the processor in a halt state, leading to a SEFI condition detected as timeout. By analyzing these faults, we found that the halt condition is the result of a fault leading to a jump into an unused code area where the custom instruction opcode is found (resulting from random initialization of the memory). The NIOS-II processor has a reserved "custom instruction" opcode to enable instruction-set extensions through the addition of custom hardware. The opcode for the custom instruction is predefined and it is not detected as illegal, even if the processor is not equipped with the custom hardware to execute this instruction. To avoid this situation we initialized the whole unused memory with an illegal opcode. In this way, any fault causing a jump to a word in the unused memory triggers an exception, and can be detected.

Once the system has been set up and tuned, we run a preliminary set of injection results, during which we injected 100,000 SEUs in the processor program counter, only. These experiments were useful for getting an initial indication of the soundness of the approach by considering very critical faults, which may dramatically harm the health of the system. We observed the following classification of fault effects:

No effect	6178
Data detection	8498
Exception	63110
Timeout	0
Trap	22161
Wrong answer	53
Total	100000

From these results we can see that 53 of the injected faults (0.053 %) escaped the detection mechanisms the architecture embeds. Moreover, by looking in more details to the results, we observed that the following traps are executed:

- 9.029 illegal opcode traps, indicating the effectiveness of properly initializing the unused memory, which is likely to allow avoiding the timeout condition;
- 4.448 misaligned data address trap;
- 8.683 misaligned destination address trap;
- 1 supervisor only instruction trap.

The results suggest that the processor already embed very powerful mechanisms for detecting misbehaviors induced by SEUs.

We also observed that SEU effects are strongly related to the position of the fault in the Program Counter. All the faults affecting bits 0 or 1 on the Program Counter (the least significant bits) were detected by misaligned memory access traps. On the other side, all the faults affecting the highest order bits of the program counter provoked accesses out of the memory regions configured on the MPU, and consequently were detected as region violation exceptions. For the intermediate range of bits, the fault effects gradually change from a majority of no effect and data detection to a majority of

illegal instruction and region violation exceptions as the affected bit varies from lower to higher order bits.

IV. Conclusions and future works

As reprogrammable FPGAs become the devices of choice for developers of safety- or mission-critical applications operating in radioactive environments suitable mitigation techniques are needed against soft errors originated in the FPGA devices. In this paper, we describe an architecture that exploits task-level redundancy in combination with already available IP cores for implementing a robust processor-based system with respect to SEUs. Preliminary results focusing on the processor program counter outline the effectiveness of the architecture, and the overhead analysis shows that the proposed architecture is effective in reducing the resource occupation with respect to N-modular redundancy, at an affordable cost in terms of application execution time. The reported results allow a deeper understanding of the fault behavior and of the effectiveness of the different fault detection mechanisms.

As future activities we intend to further validate the robustness of the architecture by performing more extensive sets of injection experiments. Moreover, we intend to implement mitigation techniques against SEUs affecting the FPGA configuration memory, and to assess the robustness of the obtained system by means of accelerated radiation testing.

References

[1] Ceschia, M.; Violante, M.; Reorda, M.S.; Paccagnella, A.; Bernardi, P.; Rebaudengo, M.; Bortolato, D.; Bellato, M.; Zambolin, P.; Candelori, A.; "Identification and classification of single-event upsets in the configuration memory of SRAM-based FPGAs," Nuclear Science, IEEE Transactions on , vol.50, no.6, pp. 2088- 2094, Dec. 2003

[2] Heiner, J., Sellers, B., Wirthlin, M., Kalb, J.; "FPGA partial reconfiguration via configuration scrubbing"; FPL 09: 19th International Conference on Field Programmable Logic and Applications, pp. 99-104, 2009.

[3] D. K. Pradhan, Fault-Tolerant Computer System Design, Prentice Hall

[4] Xilinx TMR tool, http://www.xilinx.com/milaero

[5] M. Sonza Reorda, M. Violante, C. Meinhardt, R. Reis, "A low-cost SEE mitigation solution for soft-processors embedded in Systems on Programmable Chips", IEEE Design Automation and Test in Europe, 2009, pp. 352-357

[6] M. Pignol, "DMT and DT2: two fault-tolerant architectures developed by CNES for COTS-based spacecraft supercomputers", IEEE Int. On-Line Testing Symposium, 2006

[7] NIOS-II Processor Reference Manual, Altera Corporation, San José, CA, USA, 2009